NASA/TM—2000-209634

Remote Control and Data Acquisition: A Case Study

Alfred J. DeGennaro
Cleveland State University, Cleveland, Ohio

R. Allen Wilkinson
Glenn Research Center, Cleveland, Ohio

February 2000

The NASA STI Program Office . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program Office plays a key part in helping NASA maintain this important role.

The NASA STI Program Office is operated by Langley Research Center, the Lead Center for NASA's scientific and technical information. The NASA STI Program Office provides access to the NASA STI Database, the largest collection of aeronautical and space science STI in the world. The Program Office is also NASA's institutional mechanism for disseminating the results of its research and development activities. These results are published by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA's counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or cosponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services that complement the STI Program Office's diverse offerings include creating custom thesauri, building customized data bases, organizing and publishing research results . . . even providing videos.

For more information about the NASA STI Program Office, see the following:

- Access the NASA STI Program Home Page at ***http://www.sti.nasa.gov***

- E-mail your question via the Internet to help@sti.nasa.gov

- Fax your question to the NASA Access Help Desk at (301) 621-0134

- Telephone the NASA Access Help Desk at (301) 621-0390

- Write to:
  NASA Access Help Desk
  NASA Center for AeroSpace Information
  7121 Standard Drive
  Hanover, MD 21076

NASA/TM—2000-209634

Remote Control and Data Acquisition: A Case Study

Alfred J. DeGennaro
Cleveland State University, Cleveland, Ohio

R. Allen Wilkinson
Glenn Research Center, Cleveland, Ohio

National Aeronautics and
Space Administration

Glenn Research Center

February 2000

Acknowledgments

A special thanks to the following people without whose support this effort would not be possible: Sarah DeGennaro, Kyung-Yang Min, Bob Kusner, and Ted Wright for their careful criticisms; Dan Oldham for his most recent upgrade work; the Zeno space experiment team for the original strip charting software; and Mike Bayda for the original help to convert from LabView to the current suite.

Available from

NASA Center for Aerospace Information
7121 Standard Drive
Hanover, MD 21076
Price Code: A03

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22100
Price Code: A03

Remote Control and Data Acquisition: A Case Study

Alfred J. DeGennaro
Cleveland State University, Cleveland, OH

R. Allen Wilkinson
NASA Glenn Research Center at Lewis Field, Cleveland, OH

Abstract

This paper details software tools developed to remotely command experimental apparatus, and to acquire and visualize the associated data in soft real time. The work was undertaken because commercial products failed to meet the needs. This work has identified six key factors intrinsic to development of quality research laboratory software. Capabilities include access to all new instrument functions without any programming or dependence on others to write drivers or virtual instruments, simple full screen text-based experiment configuration and control user interface, months of continuous experiment run-times, order of 1% CPU load for condensed matter physics experiment described here, very little imposition of software tool choices on remote users, and total remote control from anywhere in the world over the Internet or from home on a 56 Kb modem as if the user is sitting in the laboratory. This work yielded a set of simple robust tools that are highly reliable, resource conserving, extensible, and versatile, with a uniform simple interface.

1 Introduction

A common task of the National Aeronautics and Space Administration's (NASA) microgravity laboratories is to develop and improve scientific instrumentation supporting peer-reviewed research that may occur in either Earth- or space-based laboratories. Many of these instrument packages have been a combination of specialized hardware and custom software. Driving needs for this instrumentation are (1) remote operations capability (telescience), (2) versatile configuration with minimal software modification, (3) reusability, and (4) simplicity and familiarity of operations for scientists. Given such motivations, a moderately paced and deliberate evolution of well-designed, tested, and extensible software will be the least expensive path producing the most robust toolset.

1.1 Background

Telescience may be generally defined as the ability to provide geographically scattered researchers with viewable data and a means by which to command instruments in a near real time capacity termed "soft real time". NASA has been involved in developing telescience for decades [1]. Historically tests have been conducted within the context of controlling satellites [2]. More recently microgravity science shuttle missions have expanded telescience for remote control of many common experiments concurrently [3].

The authors have had direct experience with four microgravity telescience-based missions in the early 1990's: the International Microgravity Laboratory (IML1 and IML2) and the United States Microgravity Payload (USMP2 and USMP3). On these missions commanding was done by the ground support teams with a finite set of manually controlled scripts sent via a dedicated network to a broadcast center and subsequently transmitted to the scientific apparatus onboard the shuttle. Results were piped down from the

shuttle through the dedicated network to the scientist teams in the ground operations center where they were viewed, stored, and used in analyses. Results from the analyses allowed researchers to make numerous decisions on modifying the course of their experiment. The missions were highly successful with IML2 being heralded as the then longest manned telescience mission [4].

A valuable windfall from these microgravity related space missions was a freeware-based data visualization tool that implemented the stripcharting of live mission data. This software has been well received, has been used on several successive flights, and is still being used today. It is this kind of low-cost software reuse and remote capability that are prime motivators in this software design and provide a working model for what is possible.

1.2 Current Experiment

The present ground-based experiments employ phase shifting interferometry in the study of the thermal diffusivity of near-critical pure fluids. Figure 1 depicts the current lab configuration. This figure is abstracted to represent only those connections between devices relevant to this discussion. The computer controlling the experiment is a 200 MHz Pentium PC with 64 MB of memory, 3 GB of disk storage, a video frame grabbing board (FG), an IEEE general purpose interface bus (GPIB) card, and a network interface card (NIC). The GPIB card transfers commands to and gathers data from two precision 10-channel multimeters, and a 4-channel digital-to-analog (D-to-A) converter. The FG interacts with a free running charge coupled device (CCD) video camera with a field of view of the test cell. Images are captured in batches of 16 at intervals. The images are 540×480 8-bit greyscale pixels. The 16-bit D-to-A converter controls two very stable DC power supplies that power the heaters located in the shells surrounding the test cell. Two more channels of the D-to-A converter control the amplitude of the square wave voltage applied to the liquid crystal phase retarder (LCC).

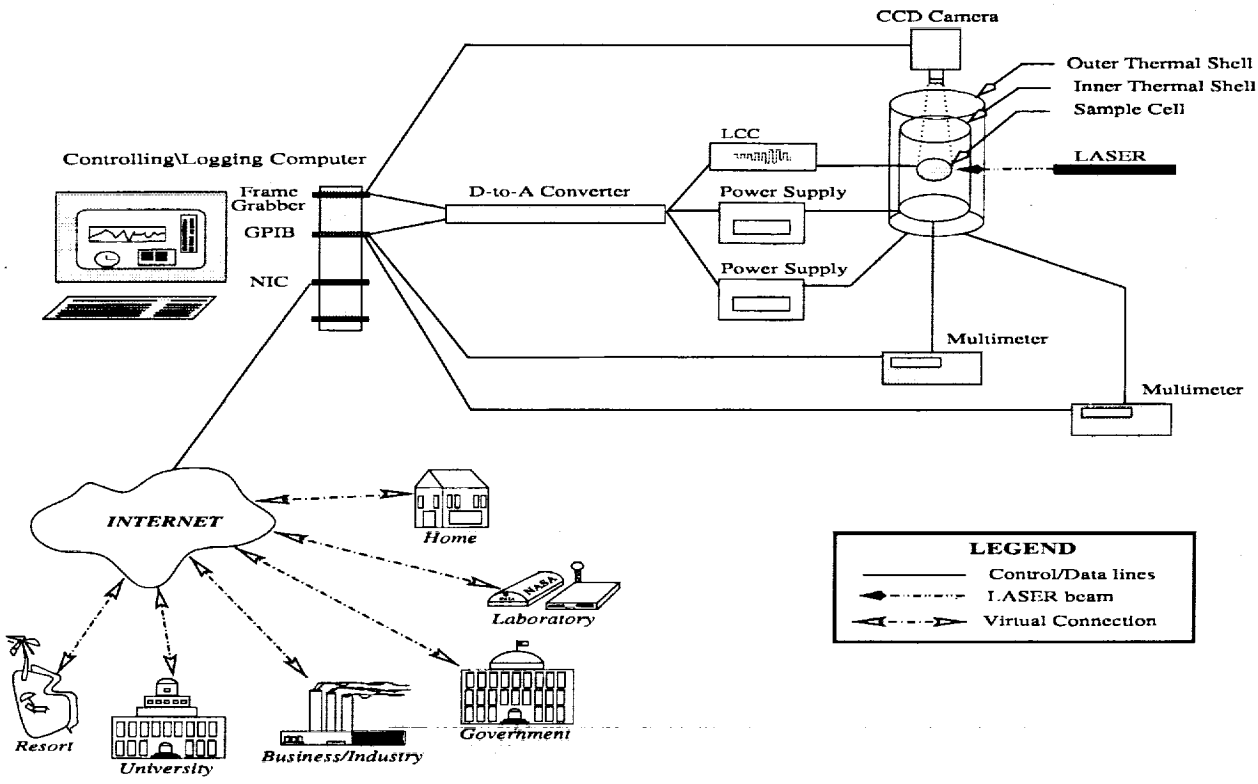

Figure 1: Lab configuration

The interferometry requires square wave amplitude control of the voltage applied to the LCC and a 5 frame per sec image capture rate to adequately measure the thermodynamic equilibration on interest. The temperature control requires active feedback proportional-integral-differential (PID) controls on two thermostat shells for ± 50 μC RMS stable temperature over 60-hr runtimes on the innermost shell. To confirm or modify experiment progress, sample temperature along with 16 other variables and captured images must be viewed during the 60-hr runs (typically over weekends). Therefore, full command and data viewing is needed over the Internet via a low bandwidth modem. One must be able to log in and out without any affect on the running processes.

In summary, there are two to four physical devices interacting with two to seven processes over days and weeks of operation. Any process can be terminated or started with any number of instances of any process at any time. Some FG operations are time critical to the 10 ms level. The computer is not heavily loaded by the processes. Remote access is identical to being in the lab at the computer even over a modem.

The remainder of this paper will describe the authors' prior experience with software control of experiments, motivations for choices with respect to robust data acquisition, specific software design and implementation paradigm, successes of this approach, and conclusions from this work.

2 Previous Solutions and Lessons Learned

The suite of data acquisition tools developed are now in their third revision. The first version was generated using Windows NT and LabView. However, the learning curve and maintenance requirements, as well as vendor upgrade frequency, exceeded frustration thresholds. This original data acquisition suite required a dedicated person committed to grooming and maintaining the system. Changes to the system, for example, adding new instrument hardware or reconfiguring channel attributes, required a rewrite and recompile of the source code. Perhaps the greatest difficulty lay in the fact that one was largely incapable of remotely accessing the system to either query the state of long duration experimental runs or to perform any commanding of instruments.

2.1 Graphical User Interfaced and Windowing Environments

GUI development tools, which promise a shorter time to move code into production, failed to produce a good fit to the scientists' expectations. Specifically, it was found that these environments were nonintuitive.

For example, GUI's hide the "how" of software execution from a researcher. That is contrary to the research instinct. Note that when tool design fails "to provide good conceptual models" or "make things visible", then humans have a difficult time in understanding how to use the tool and fail to discover all possible functions that may exist [5].

> "Technology offers the potential to make life easier and more enjoyable; each new technology provides increased benefits. At the same time, added complexities arise to increase our difficulty and frustration. The development of a technology tends to follow a U-shaped curve of complexity: starting high; dropping to a low, comfortable level, then climbing again. New kinds of devices are complex and difficult to use. As technicians become more competent and an industry matures, devices become simpler, more reliable, and more powerful. But then, after the industry has stabilized, newcomers figure out how to add increased power and capability, but always at the expense of added complexity and sometimes decreased reliability." [5]

Experience has demonstrated that GUI programs grow much as flowcharts do. Flowcharts; however, failed to be used effectively because they grew without bounds in all directions. GUI programs, too, may spread over a large number of pages and be several layers of instructions deep. Many times these programs are nonplanar requiring control lines to cross each other over and over again. This tends to create a mesh which is, at best, difficult to follow. It is reminiscent of "spaghetti code" [1.6], which was a result of programmers abusing the unconditional branch statement within their programs. The result is that production grade

GUI programs written in windowing environments are often difficult to modify, maintain, and difficult to demonstrate reliable.

It is the authors' view that there is a lack of clear design principles for GUI programming. Without clear design principles one finds oneself in a situation much as with early programming attempts, that is, programmers are pushed into a mind set of "code and fix" [6]. This model fails to be effective as it often leads to poorly written code that is expensive to maintain. Worse still such coding paradigms lead to code that is almost impossible to demonstrate as correct and hence unreliable [6]. If circuit design methods could help to control some of the design issues then it would be a great merging of what has traditionally been two disparate groups. Unfortunately, too few programmers have the experience with circuit design techniques to adequately design within a GUI programming environment. While significant strides are being made [1, 7] these have yet to become widely adopted.

In summary, the authors found that the unneeded complexity of the GUI interface and underlying code should be avoided. This made software development quicker with higher performing code while keeping the user interface simple and more predictable.

2.2 Commercial Off-The-Shelf (COTS) Software

Commercial data acquisition, plotting, and image viewing and processing packages under Windows-XX or MAC-OS suffer from being proprietary with some excessive licensing schemes. As such, details of critical algorithms are kept out of the reach of researchers. When this proprietary black box approach becomes a strategy for defining a market niche, it severely hampers the ability of the researcher to understand the data being presented by the software. Unexpected results are common in pioneering science. When a researcher sees unexpected results, then all elements of the data collection are re-examined. Any hidden elements block the research progress. Commercial software with its hidden functions block research.

Software that supports instrumentation is rarely sufficient in its present state. Constant improvement is sought. To do this researchers need the ability to change current software functions, but only in areas of their choosing. It is tedious to be caught in the vendor loop of constant upgrades and compatibility issues for functional upgrades of no interest.

Commercial software support for sharing scientific results via teleconference has been limited. Commercial software usually has bloated proprietary data formats that slow data transmission and require the same software by everyone. Historically, only UNIX systems have allowed reasonably secure live data sharing over the Internet. Recent open-source UNIX systems have enabled easy low-cost data exchange. Readily available source code and algorithms, efficient software, graphical functions as needed, data formats that all can read and view, and mature tools motivated by function and not marketing, illustrate some of the richness that this environment provides. Open-source software gives the researcher much that commercial software does not.

2.3 Operating System Choices

Real time issues are crucial to good experiment control. Yet, as obvious as this is, most GUI environments do not allow users to have complete control over critical timing issues [1]. Screen updates, mouse motions, and disk updates can interfere with the user application. The workaround is to set the data acquisition machine aside so as to be usable only by the experiment. However, this solution is extreme for many purposes. Much work is being conducted in this area with the result that many of the most popular operating systems (OSs) are recently providing a form of soft real time environment [8–13]. As expected, some OSs accomplish this functionality more readily than others. In particular, UNIX environments [14] offer a strong solution as they (1) allow priority setting by users, (2) provide robust, reliable, secure Internet remote access, and (3) support "mature" multitasking [9, 10, 12, 15–17]. Others do not offer these nearly as well or completely. They are forever making half steps towards such features at the expense of users.

2.4 Custom Software

Custom software can be costly, often nonintuitive, frequently characterized as a dedicated solution, and often not reusable without being deliberately targeted for that at inception. If done poorly it is difficult to modify and maintain, and often ties a researcher to a particular programming group. Solutions done "in house" frequently lack the more sophisticated GUI interfaces but benefit from being nearer the problem and having the ultimate consumer of the product near at hand to guide development. While solutions "contracted out" may sport the more elegant user interface but require much more energy to keep the communication channels open between developer and researcher to ensure satisfaction is achieved. The authors' preference is to develop solutions in house so as to have a much stronger understanding of the processes and their implementation. Cost is kept low by the simplicity of code and the utilization of mature software tools already developed.

Historically, the approach most often taken has been to determine what functions are required from the available devices, learn how to communicate with those devices, learn the syntax of the commands unique to each device, sort through the myriad options and possible permutations of command syntax, pick a handful of useful instructions and associated parameters, embed them in the software and release it into production. This "typical" approach suffers from a number of difficulties not the least of which is the tight coupling of instrument commands within the source. The result is that if new functions are required, a reworking of the program source is also required. Frequently this is seen as too difficult. What is more likely to be done is that some parameters for the preselected instructions are allowed to be dynamically set but the full instrument capability is usually lost.

GUI software and real time operating systems are still maturing. Currently, no one offering is universally better than any other. Software that is expected to be more than a "quick and dirty" solution will be custom designed incorporating tools that have a history of reliable performance. Yet custom software will provide the best fit to the scientists' needs only if carefully managed through to completion. As researchers' expectations of software mature software specifications will become more forthright. Developers need a disciplined approach that provides a solid design paradigm, involves the researcher throughout the development process, and also is sensitive to the researchers' perceptions of function, utility, and learning styles.

3 Critical Factors Driving Software Specifications

As a direct consequence of this history, six key factors governing software design and programming methodology have been identified. The first factor focuses on software **design and implementation** requiring that the software be sufficient to accomplish the tasks required, easy to modify, and easy to maintain. Moreover, the software must be fully open to the user at any level. The method for adding extensions to the software must be clear, repeatable, and stable. There must also be provision made for error recovery and critical system failures, that is, there needs to exist a high degree of "fault tolerance." A second factor focuses on **reliability and performance** of the software requiring that it can run for indeterminant times with any frequency of user interaction. It also requires responsive and robust multitasking. Third, **command and control** requires that users must be able to initiate, terminate, and alter the behavior of processes and devices **"on the fly"**. The interface must be familiar to the researcher, using natural metaphors. Fourth, the visualization system must provide sufficient and clear information so as to allow its users to **ascertain its state at a glance**. This is related partly to data visualization and partly to command status behavior. Fifth, the system must be **operable remotely** by many users "simultaneously" without conflict or unnecessary delay. This facility should not require excessive communications resources and should accommodate users with relatively low bandwidth connections. Finally, sixth, the system must **log data that is usable on diverse computing platforms** remotely or locally and within a variety of third party analysis software without rework. The following sections address these factors in more detail.

3.1 Software Design and Implementation

The designers need to think of creating tools that will have a useful life expectancy in excess of 10 years before major changes are required. Design must be done with a foreknowledge that systems developed will be required to (1) be extensible by multiple developers, (2) be readily adaptable to new instruments and technologies as they become needed, (3) have a stable user interface, and (4) interact with devices and functions in repeatable ways to avoid user re-learning curves. Since multiple programmers will be involved in maintaining and enhancing this software suite, documentation must be done concurrent with program development. It is widely accepted that good design for usable systems is best done in conjunction with such writing [18]. In addition, authoring of source code should make use of existing programming paradigms that stress decomposition of complex problems into simpler modules, prototyping, walkthroughs, and encourage frequent user involvement [6]. The outcome will be to produce code that is readily comprehensible and reliable, and to improve the likelihood of catching errors early in the software development lifecycle.

Specifically, this factor requires that the software be designed such that the addition of new instruments does not require a rewrite of existing, working software. Time to incorporate new instruments into experiments must be minimized. To achieve these goals one must disentangle the instrument's command language from being "hard coded" into the controlling software. To allow facility for additional (possibly unknown) instruments a small scripting language with instrument configuration capability must be designed into the solution. This scripting must be readily comprehensible by users. It must be sufficiently open-ended to allow for an arbitrary number of variable length commands to be stored and ultimately issued to the corresponding devices. In this instance it implies allowing for the reconfiguration of instruments even while the software is running.

3.2 Reliability and Performance

In the current configuration the system runs for weeks and months at a time with 3-day interferometry sequences mingled in. Data log files must be protected from computer hardware failure. Event timing must be deterministic once the controlling software receives a command. Process prioritization must allow the key processes to supersede most operating system processes (e.g., mouse movement). Process CPU cycles should be very small so that running experiments may be controlled and data viewed with perhaps 10 to 20 percent CPU load even on low-end machines (e.g., 80486 processors). Multiple processes will start, stop, and execute at any time therefore they must neither interfere with nor overrun each other. One should be able to run multiple experiments or other tasks on the computer without system failures or loss of execution determinism in the experiment. The system must not allow temperature or any other critical process to "run away." The system must be capable of automatically shutting down under such conditions. This is especially true if this system is to be used on shuttle or Space Station missions.

The system will be implemented in stages with testing being performed at multiple levels. The programmers will be responsible for programming defensively, incorporating frequent incremental checks, employing "test coverage", etc. [19]. The researchers will test the functionality and system performance. Finally, if the software moves on to Space Station science use it will undergo numerous simulations and bounds checking.

3.3 Command and Control

This software suite should separate processes running concurrently in a multitasking environment, each controlling its own device [20]. Processes will need to communicate with each other as well as be interfaced with the user. Users anticipate the need to be able to change the setup profiles of instruments while the experiment is running and to do so without restarting the experiment. Similarly runtime scripts that control instrument execution may be changed by the user at any time. This enables starting a new experiment from an already running experiment.

The command interface must be simple and intuitive [21, 22]. Instruments should be able to be started and terminated remotely. They should survive logoff of any terminal session. In fact any process should be able to be started or terminated without adversely impacting any other running process. The interface may

be replicated by multiple users in various locations and should reflect the current status to all users logged in simultaneously.

3.4 Data Visualization and Experiment Status

Data values are usually viewed as a function of time and sometimes as a function of each other. Strip charts do this well. Images are the biggest consumer of computer resources and must not be biased by any processing for storage (e.g., lossy versus lossless). In all cases live data graphics are preferred over tables and blinking numbers. This work to date has not had to handle animated surfaces or live video. There is nothing in this software that prohibits that future capability. Another requirement is portability of data files between OSs and applications. A particular strength of portable data log files is that one is not limited in choices of display software. Stripchart display tools should be allowed or enabled either on the remote terminal or on the lab host based on user choice. Process command and control interfaces should be constrained to full-window text-based tools to improve communication performance and enable simplicity with extensibility for remote and local users alike.

3.5 Remote Operations Capability

This work has involved multinational users. It is a goal to provide members of any research team from around the world the ability to access and interact with what is done in the lab. Furthermore, it is intended that these systems be designed as a means for potential education outreach. Systems such as these could become a mechanism by which school systems find ways to involve students in research projects using equipment and facilities that they could not afford to own themselves. Ultimately this software could be used to control experiments on either the Shuttle or Space Station.

With such a diverse group of users, it is clear that the command and viewing tools need to be simple and familiar. The bandwidth needed for interaction should be manageable with a modem. Data should be portable in an open format. Specifically, design will be implemented around existing communications media. The choice here is an Ethernet and the TCP/IP protocols between remote terminals and in-lab host control computers. This is a familiar technology with much work being done to improve throughput and enhance reliable data transfer. The intent is to leverage such foundational tools throughout the design.

3.6 Data Logging and Data Formats

There are two types of data recorded here, images and measured transducers over time. To store this data in the most portable and efficient format, raw pixel format and text files are used respectively. File sizes are on the order of 10 MB for text logs and 0.25 MB per image for pictures. Efficient data transmission is left to the user's default hardware and software. For example, a home PC with a 56 Kb modem logged on to the lab host using secure shell (ssh) permits commanding and stripcharting with adequate performance. Proprietary encrypted formats are avoided because of nonuniversal software tools needed to access the data and the excess size of such formats.

4 Natural Data Acquisition With Remote Operations

The problem was divided into four parts: (1) to establish clearly defined communication areas in shared memory for each device that must be managed, (2) to run each device as a separate concurrent process, (3) to provide a means to monitor and control devices independently and remotely, and (4) to provide data visualization tools to monitor progress of the experiment locally or remotely. The model, illustrated in Figure 2, consists of three sections identified from left to right as "initialization", "shared memory" to support interprocess communication (IPC), and "concurrently running processes".

The first tool named "EXPeriment" (EXP) is used to create and load the communication areas into a shared memory space. Instrument commands are read from setup scripts stored in an initialization file (see

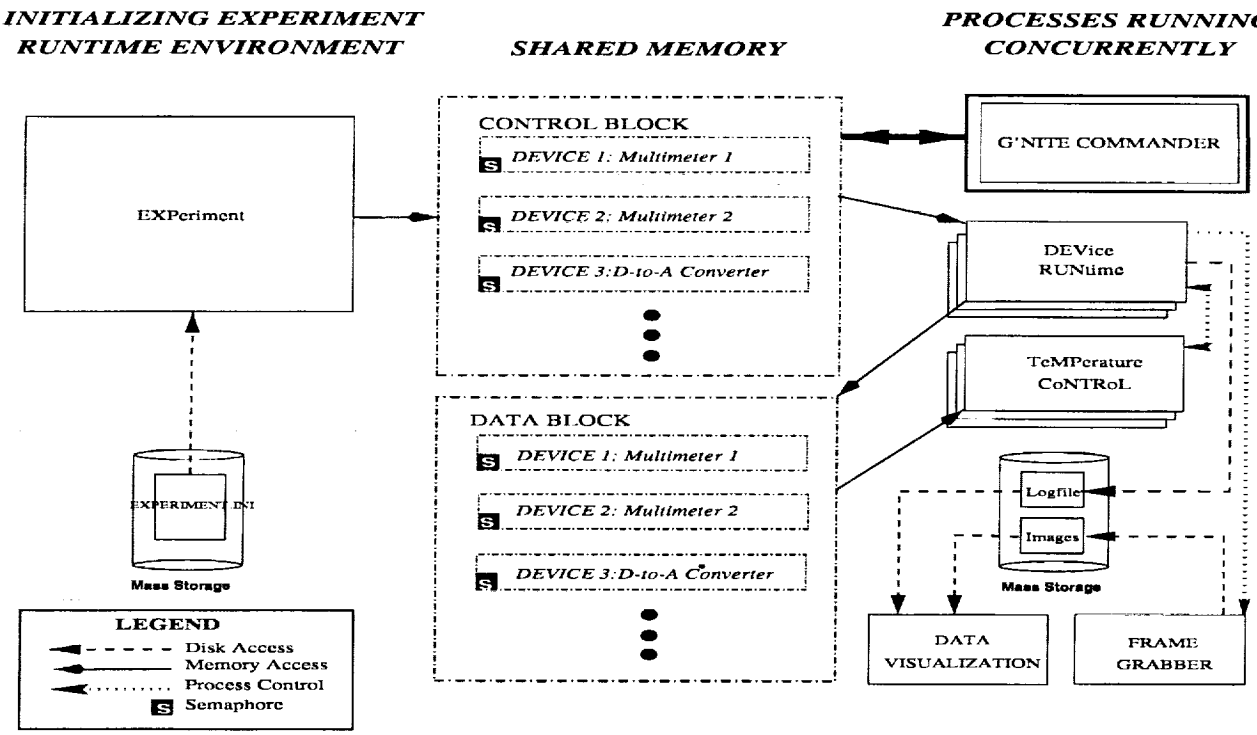

Figure 2: Software Model. The suite is identified by EXP and DEVRUN. G'NITE is the command interface. TEMPCNTRL, Frame Grabber, etc. are examples of the extensibility of this software suite.

Appendix for a sample .ini file) and written to the shared memory segments located in the CONTROL BLOCK (see Fig. 2) one set per device, for example, *multimeter1*. The user needs to learn the few command strings that make up the scripts, and the command strings that are unique to each instrument. Researchers want to know such command strings if an instrument has routine utility and quality computer-based measurements are needed. The scripts are text files based upon a simple definition syntax, which allows users to specify instrument initialization, process control, and instrument termination command sequences unique to each device. As denoted in the figure this is the only process of the suite that is allowed to *write into* the CONTROL BLOCK region of shared memory. The EXPeriment module also attaches what are essentially virtual instruments to physical instruments by registering GPIB addresses as assigned by the user. This program need only be run once at computer power up. However, it can be re-run at any other time to reconfigure any of the instruments, even while an experiment is running.

After the shared memory is established and instructions "uploaded" the DEVice RUNtime (DEVRUN) packages are started; one for each instrument. These programs *read from* the shared memory CONTROL BLOCK to set up and command their corresponding devices. A state diagram of the DEVRUN program is detailed in Figure 3. Multiple instances of DEVRUN may be run each in one of two modes. The first and simplest mode is described in the leftmost portion of the figure and depicts a typical session. Observe that once a process is begun it will drop into an infinite loop (signified by the dotted arrow) until either commanded by the monitoring process to quit or a PANIC instruction is received. The loop is executed on a user specified time interval and follows the sequence; command device, read data, (optionally) filter data, open logfiles, log data, close logfiles, and SLEEP until signaled to repeat loop. Data logs (see Fig. 5) are generated by each instance of DEVRUN with the relevant column headers defined for each device in its respective CONTROL BLOCK. Opening and closing logfiles with each DEVRUN loop is a way to preserve data upon computer or power failure. Periodically, new logfiles are started at the user's request without the need to restart any processes that make up the experiment.

As suggested above, the DEVRUN modules may also be modified to do some processing on data that has been acquired. This feature is provided through a collection of *filters*, one for each DEVRUN, made active through command line arguments used at execution time. The filters are written in "C" and compiled into the DEVRUN modules themselves.

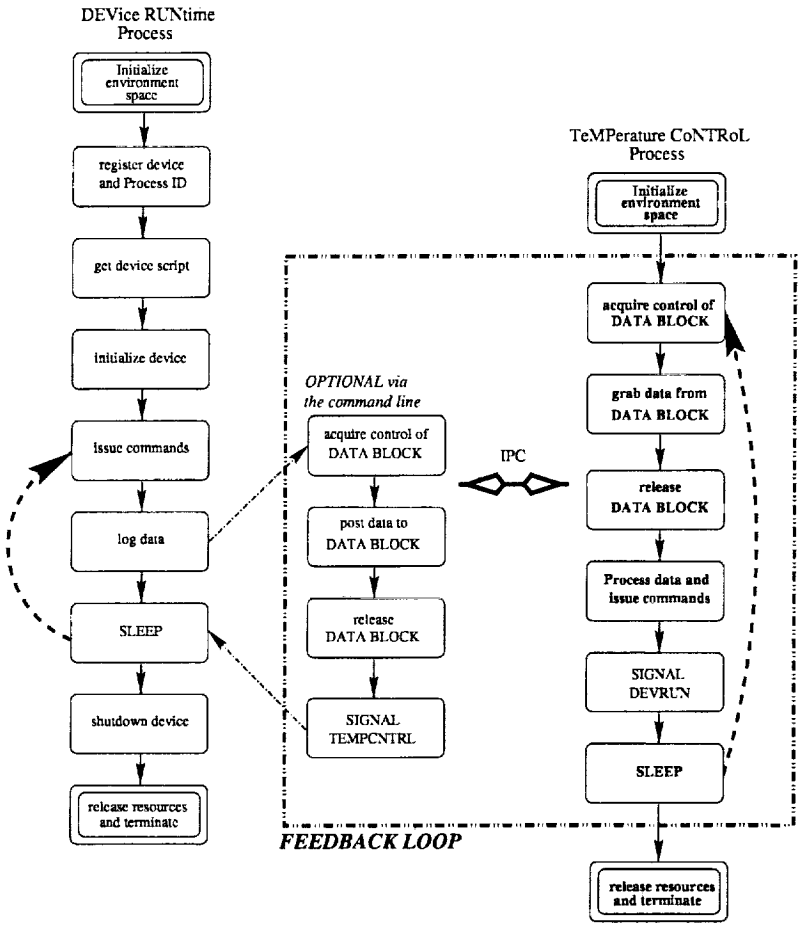

Figure 3: State Diagram detailing the DEVice RUNtime process. Also depicted is an optional feedback loop featuring an InterProcess-Communication (IPC) interface between two concurrently running processes; DEVRUN and TEMPCNTRL.

As an alternative, DEVRUN may be run in a second "POSTING" mode. This mode allows DEVRUN to be used in conjunction with other concurrent processes, for example, temperature control. In this way *feedback loop control* is provided. When invoked in this fashion the standard sequence of events is expanded (see Fig. 3) so that the DEVRUN process will *write* or "post" its results to both the logfiles and to another portion of shared memory referred to as the DATA BLOCK. Providing controlled access to this sensitive data area is crucial. To ensure data integrity and avoid DATA BLOCK overruns, semaphores have been employed [23]. Processes must first query the condition of the appropriate semaphore before using the related DATA BLOCK segment of shared memory. After posting its data, DEVRUN will relinquish control of the DATA BLOCK, signal its peer process and then sleep until the next cycle. The corresponding process will then attempt to acquire the DATA BLOCK, grab the data, release the data segment, process the data, command any devices as necessary, and then wait until it is signaled that new data is posted at which point the cycle begins anew. It is in this way that additional complex processing may be implemented.

The TEMPerature CoNTRoL modules are an excellent example of how this system may be extended to incorporate additional user functionality. InterProcess-Communication (IPC) is accomplished via signal interrupts and shared memory as described above. Once data is posted by DEVRUN, a user module may do whatever it requires without directly impacting the data acquisition suite. In this work, the suite was extended to do PID temperature control on two thermostat shells each with its own instance of TEMPCNTRL.

Other examples of extending the system are the FG module that manages the image capture and LCC. Likewise, an oscilloscope device has been added. The oscilloscope is implemented as a DEVRUN instance, while the FG is a modification of vendor supplied DOS source code to run in the LINUX environment. The FG process sleeps until it is triggered manually or by a timer. Upon receiving the *trigger command* the program steps through a series of voltages capturing rapid images of the test cell. The FG process then returns to sleep completing the cycle.

For stripchart data visualization, previously developed open-source space experiment software has been used (see Fig. 5). The software was developed with mature Athena widget and X-window tools. Image data are 540×480 8-bit grey scale pixels, stored in a raw pixel format, and accessible by a number of visualization and analysis tools such as ImageMagick, PV-WAVE, IDL, or MatLab. Image viewing tools may be chosen to suit user preference.

Remote operation is via simple telnet or secure shell logins. All processes can be started, interrogated, commanded, or terminated during a login session. Logoff can occur with no impact on running processes. Data visualization requires X-Windows software on the remote terminal. X-Windows for visualization was chosen because of its availability on all OSs, open standard based, and maturity. This suite is low bandwidth such that a 56 Kbps modem handles all but strip charts and images instantly. Secure Shell's compression scheme enables graphics refresh on the order of 10-15 seconds. Remote interaction is therefore fully functional as if sitting at the laboratory computer.

4.1 Interprocess Communications (IPC)

An important feature to note in this design is that memory areas are partitioned to force a type of data integrity (see Fig. 4). Only the EXP process is allowed to write into the CONTROL BLOCK region of shared memory. The DEVRUN processes may only read from the CONTROL BLOCK. When communication with user defined processes (e.g., TEMPerature CoNTRoL) is implemented through what is in essence a virtual bulletin board (i.e., the DATA BLOCK), adhering to the use of semaphores for posting and acquiring data helps to assure that other processes will be well behaved in this common data region. The DEVRUN processes themselves treat data in the DATA BLOCK as untrustworthy and hence log only the data acquired directly from the devices controlled. Processes that use the data from the DATA BLOCK must do their own data validation. If other processes somehow corrupt this region it will not affect the data acquisition suite's reliability. It is the responsibility of the developers making additions to the suite to prevent violating this read/write rule and avoid potentially disastrous consequences.

4.2 The Command Interface

G'NITE Commander (see Fig. 5) is the name given to the device command user interface. It was developed using the ncurses library which allows for complete text-based screen management in a UNIX environment. The interface itself is rather spartan with the screen divided into several parts: (1) display, (2) command line, and (3) function key descriptions. Context sensitive help on control command syntax is also provided. The display allows one to view the various states of running processes or memory as requested by the user. Users may select to view process states, write new device commands directly into the CONTROL BLOCK, view or alter the DATA BLOCK for parameters like set-points, or terminate processes directly. This is the only tool that may actively alter any of the shared memory space and should therefore be used with prudence.

This interface requires a user to realize the shared memory blocks concept and know the command strings or parameters each device understands. The rest is self-explanatory.

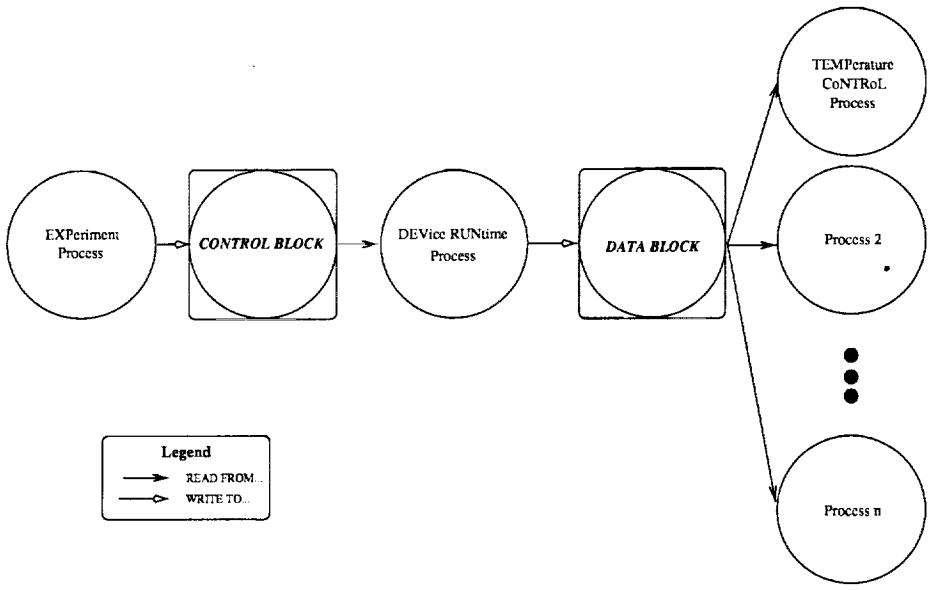

Figure 4: Data integrity forced by direction of writes and reads.

5 Measures of Success

The current system has been in place for two years successfully controlling the interferometry experiment. The system is remotely controlled via a text-based user interface that allows command of all aspects of the experiment environment. The system has run continuously for extended periods of time (usually several months) without either restarting the suite or rebooting the computer. The CPU usage is less than 1% for all but strip chart graphics updates, which peak at 3.3%, for the interferometry experiment. One can therefore run multiple experiments on one computer and GPIB interface such as running several interferometer experiments and acquiring data from an oscilloscope or laser power meter interactively.

Additions to the suite have been made since its original development. These enhancements have been done seamlessly and independently of the core suite. Instruments have also been added without necessitating a rewrite of the code. Instruments may be installed at runtime or added later while the experiment is running without adversely impacting the runtime environment.

Using an Open Source platform has allowed researchers to incorporate additional features into the environment without modifying existing tools. For example, the use of Secure Shell improves security without debilitating the suite in any way. Quite the opposite, with the compression features such tools offer researchers improved remote performance.

Designing the system as loosely coupled modules with weak cohesion, involving users in design issues, and purposing to create code that is maintainable has generated a system that is simple, robust, and efficient and was done so in a remarkably short period of development time. The entire system (see Table 1) was developed over a 10-week period. Eight nonfatal errors were found and were addressed in subsequent patches. The system has been tested rigorously and continues to be used as a daily research tool.

6 Conclusions

One of the key features of this approach is the ability to separate commands unique to each instrument from being embedded into the source code. This markedly reduces the time it takes to bring instruments into the context of the experimental environment as users need not focus on coding issues but only on how to set up and command their instrument(s). Separation of tasks means that the system is more reliable overall.

Module Name	Number of Lines of Code (LOC)	Number of Blank Lines	Number of Comment Lines	Number of Comments
EXPeriment	492	54	66	69
DEVice RUNtime	443	53	194	147
TEMPerature CoNTRoL	363	50	70	73
G'NITE Commander	693	41	23	13
Manual Trigger	41	7	19	21
Filter A	27	19	21	22
TOTAL	2059	224	393	345

Table 1: Program development metrics for the Data Acquisition Suite and the G'NITE Commander

Allowing for independent processes to be run concurrently and providing communication via IPC hooks (i.e., interrupts and the DATA BLOCK) implies that new programs may be added without rewriting the existing systems. So long as the directed method for reading and writing shared memories is not violated one may have confidence that processes will run reliably. Restricting data logs to generic open formats grants a wider range of tools from which to select for live visualization and post processing of data. Having written these applications in ANSI C may shorten the time required to port them to other platforms, especially other UNIX's. Serendipitous advances in open source UNIX's will serve to make this solution more attractive to users.

This work confirms the belief that the six critical factors of scientific software: (1) careful systematic design and implementation, (2) reliable performance, (3) minimal control mechanisms, (4) simple data visualization, (5) remote capability, and (6) portable data formats, are achievable with modest resources and will produce high-performance, high-quality software tools that will find a value beyond their immediate use. This suite provides a natural, transferable medium through which to conduct experiments. This effort has not sought to exhaust all of the possible tool types that one may value in conducting research. More importantly this work in no way impedes such growth but rather provides a solid bedrock of tools from which to do further development. It is puzzling that this approach, which stresses simplicity, clarity, and extensibility of laboratory software, seems to be foreign to software developers and downright alien to vendors as a whole.

References

[1] J. R. Matey. A world without barriers: Editor's report from NIWEEK '97. *Computers in Physics*, 11:570ff., Nov/Dec 1997.

[2] American Institute of Aeronautics and Astronautics. *Telescience at the University of California, Berkeley.* 39th Congress of the International Astronautical Federation, October 1988. Space Sciences Laboratory, University of California, Berkeley, CA 94720.

[3] R. Schuiling. Telescience is put to the test. *Spaceflight*, 35:68ff., 1993.

[4] Roelof Schuiling. Longest flight of the space shuttle program: International Microgravity Laboratory-2 demonstrates telescience. *Spaceflight*, 1994.

[5] D. A. Norman. The psychopathology of everyday things. In W. A. S. Buxton R. M. Baecker, J. Grudin and S. Greenberg, editors, *Readings in Human-Computer Interaction: Toward the Year 2000*, page 5ff. Morgan Kaufmann Publishers, Inc, San Francisco, California, 2nd edition, 1995.

[6] B. W. Boehm. A spiral model of software development and enhancement. In et al. R. M. Baecker, editor, *Readings in Human-Computer Interaction: Toward the Year 2000*, page 281ff. Morgan Kaufmann Publishers, Inc, San Francisco, California, 2nd edition, 1995.

[7] T. Williams. It takes more than a keen nose to track down software bugs. *Computer Design*, 32:67ff.–, September 1993.

[8] R. Grehan. NT in real time. *Byte*. 21:86NA3–, 1996.

[9] Moses Joseph. Realtime POSIX: Boon or bunk? *Computer Design*, 33:156ff.–, September 1994.

[10] D. Hildebrand. POSIX for realtime embedded systems. *Computer Design*, 34:136–, 1995.

[11] J. Challenger. Visual programming for realtime. *Computer Design*, 33:120–, August 1994.

[12] M. Barabanov and V. Yodaiken. Real-time UNIX. email:yodaiken@nmt.edu.

[13] W. Oehme and S. Brosky. High times for realtime computers. *Machine Design*, 65:44ff.+, March 1993.

[14] T. Williams. WindowsNT challenges UNIX for embedded and realtime development. *Computer Design*, 33:47ff.+, 1994.

[15] T. Williams. Tools help preserve propriatary realtime system software investment. *Computer Design*, 32:36–, August 1993.

[16] T. Williams. Libra frees embedded programs from priority scheduling. *Computer Design*, 34:36ff., 1995.

[17] J. Park and Y. Yoon. An extended TCP/IP protocol for real-time local area networks. email:jhyun, treeman@rcsl.inha.ac.kr, 1997.

[18] John D. Gould and Clayton Lewis. Designing for usability: Key principles and what designers think. *Communications of the ACM*, 28(3):300–311, March 1985.

[19] Brian W. Kernighan and Rob Pike. *The Practice of Programming*. Addison-Wesley, Reading, Mass., 1999.

[20] M. Burke et. al. E787 data acquisition software architecture. *IEEE Transactions on Nuclear Science*, 41(1):131–134, February 1994.

[21] Brad A. Myers. State of the art in user interface software tools. In W. A. S. Buxton R. M. Baecker, J. Grudin and S. Greenberg, editors, *Readings in Human-Computer Interaction:Toward the Year 2000*, pages 323–343. Morgan Kaufmann Publishers, Inc., San Francisco, CA, 2nd edition, 1995.

[22] A. Marcus. Principles of effective visual communication for graphical user interface design. In W. A. S. Buxton R. M. Baecker, J. Grudin and S. Greenberg. editors, *Readings in Human-Computer Interaction: Toward the Year 2000*, page 425ff. Morgan Kaufmann Publishers, Inc, San Francisco, California, 2nd edition, 1995.

[23] W. Richard Stevens. *UNIX Network Programming*. volume 2. Prentice Hall, PTR, 2nd edition, 1999.

8 Appendices

This section provides snapshots of files and images presented in the reading.

Algorithm *INSTRUMENT.INI*

```
# # # # # # # # # # # # # # # # # # # # # # # # # # #
#SAMPLE SETUP FILE FOR EXPERIMENTAL DAQ AND TEMPERATURE CONTROL (PID)
# # # # # # # # # # # # # # # # # # # # # # # # # # #
@DEV_COUNT
1            ;number of devices connected to the NI488 & setups to follow
#*****************************************************************

@DEVICE
$NAME "meter1"
$LOG  "meter1.log"
$NUMCHANNELS 6    !specify the no. of NON-comment channels to later define
$FORMAT "rc"     ! rEAD,cHAN'
$TIMER(10s0)     ! (SECONDSsMILLISECONDS)
$PRIORITY(-20)

$TITLE "TentAir(C)"      ! ch1
$TITLE "SCUWall(C)"       ! ch2
$TITLE "Flange(C)"       ! ch3
$TITLE "Pres(mbar)"      ! ch4
$TITLE "Dummy"           ! ch5
$TITLE "AHmd(g/m3)"       ! ch6
#$TITLE "c7"            ! ch7
#$TITLE "c8"            ! ch8
#$TITLE "c9"            ! ch9
#$TITLE "c10"           ! ch10
$TITLE "SetPtI(C)"       !title for COMMent field 1
#$TITLE "comment 2"      !                    2
#$TITLE "comment 3"      !                    3
$COMMENT " SetPoint"     !                    1
#$COMMENT " **."         !                    2
#$COMMENT " **."         !                    3

$SETUP
_IBCLEAR
"*RST"
":FORM:ELEM READ, CHAN"   !specify 'reading,channel' output
":INIT:CONT OFF"          !'ON' or 'OFF' to control triggering reads
":ABOR"                   !abort current going acquisition
":SYST:AZER:TYPE NORM"    !AUTOZERO MODE 'NORM' or 'SYNC'
":SYST:AZER:STAT OFF"     !NORM is faster than SYNC
":SYST:LSYN:STAT ON"      !Line Synchronization of Measurements, less noise

#":SENS:VOLT:DC:RANG:AUTO ON"      !
":SENS:VOLT:DC:RANG:UPP 5.0"      !*USER MUST DECIDE RANGE VALUE

#":SENS:RES:RANG:AUTO ON"      ! see the adjustable fixed ranges below

#***All of the following commands may change on the fly****

":SENS:RES:NPLC 2"          !*min 0.01 to max 10, (2->7.5 DIGITS)
":SENS:RES:OCOM OFF"        !*ON or OFF (ON for =<, 200k)

#    the following ranges correspond to full scale 2k, 20k, 200k,  2M ohm

":SENS:RES:RANG:UPP 1000000" !*set upper range  1000,10000,100000,1000000
":SENS:RES:AVER:TCON REP"    !*REP or MOV
":SENS:RES:AVER:COUN 14"     !*integer 1 to 100
":SENS:RES:AVER:STATE ON"    !

#":SENS:RES:RANG:UPP?"       ! diagnostics
#_SLEEP(1)
#_IBREAD

":SENS:VOLT:DC:NPLC 2"        !* min 0.01 to max 10
":SENS:VOLT:DC:AVER:TCON REP"  !* REP or MOV
":SENS:VOLT:DC:AVER:COUN 8"    !* integer 1 to 100
":SENS:VOLT:DC:AVER:STATE OFF" !

":SENS:VOLT:AC:NPLC 10"        !* min 0.01 to max 10
":SENS:VOLT:AC:AVER:TCON REP"  !* REP or MOV
```

```
70.  ":SENS:VOLT:AC:AVER:COUN 8"    !* integer 1 to 100
71.  ":SENS:VOLT:AC:AVER:STATE OFF" !
72.  # ****************************************
73.
74.  ":ROUT:SCAN:LSEL NONE"              !disable all scanning operations, clean up
75.  ":INIT:CONT OFF"
76.  ":ABOR"                        !* if there is an acquisition going, stop it
77.  ":ROUT:SCAN:INT:FUNC (@1:10), 'NONE'" !*if any channels configed, clear them
78.
79.  #  channel 1
80.  ":ROUT:SCAN:INT:FUNC (@1), 'VOLT:DC'" !*
81.  #  channel 2
82.  ":ROUT:SCAN:INT:FUNC (@2), 'RES'"    !*
83.  #  channel 3
84.  ":ROUT:SCAN:INT:FUNC (@3), 'RES'"    !*
:
:
85.  ":ROUT:SCAN:LSEL INT"           ! Select the Above Configured Channels
86.
87.  # Set up triggering protocols
88.  ":ARM:LAY1:COUN 1"         !* No. of passes thru Layer
89.  ":ARM:LAY1:SOUR IMM"       !* trigger immediately when commanded
90.  ":ARM:LAY2:COUN 1"         !*
91.  ":ARM:LAY2:SOUR IMM"       !*
92.
93.  #Dynamic and dependent on # channels active
94.  ":TRIG:COUNT 5"            ! No. of channels to scan
95.  ":TRIG:SOUR IMM"           ! trigger immediately when commanded
96.
97.  #Trace scan channels and provide/sense all readings in engineering format
98.  ":TRAC:CLEAR"          !
99.  ":TRAC:EGR FULL"       !
100. ":TRAC:POIN 5"         ! No. of channels to scan (buffer size)
101. ":TRAC:FEED SENSE"     !*
102. ":ABORT"               !
103. ":TRAC:FEED:CONT NEXT" !* rearms multimeter
104. ":INIT:IMM"            !* start multimeter to sample
105. _SLEEP(8)              !allow time for multimeter to accept and act on cmds
106. SEND
107. # ************************************
108.
109. $RUNTIME
110. ":SYST:AZER:STAT ON"   ! take a little dead time to reset zero on DMM
111. ":TRAC:DATA?"          ! send data from buffer to computer
112. _IBREAD                !
113. ":ABOR"                !
114. ":SYST:AZER:STAT OFF"
115. ":TRAC:FEED:CONT NEXT" ! rearms multimeter
116. ":INIT:IMM"            ! triggers multimeter again
117. # go off and do pid after this statement and log to file here
118. SEND
119. @D_END
120.
121. @EOF                !end of setup file
```

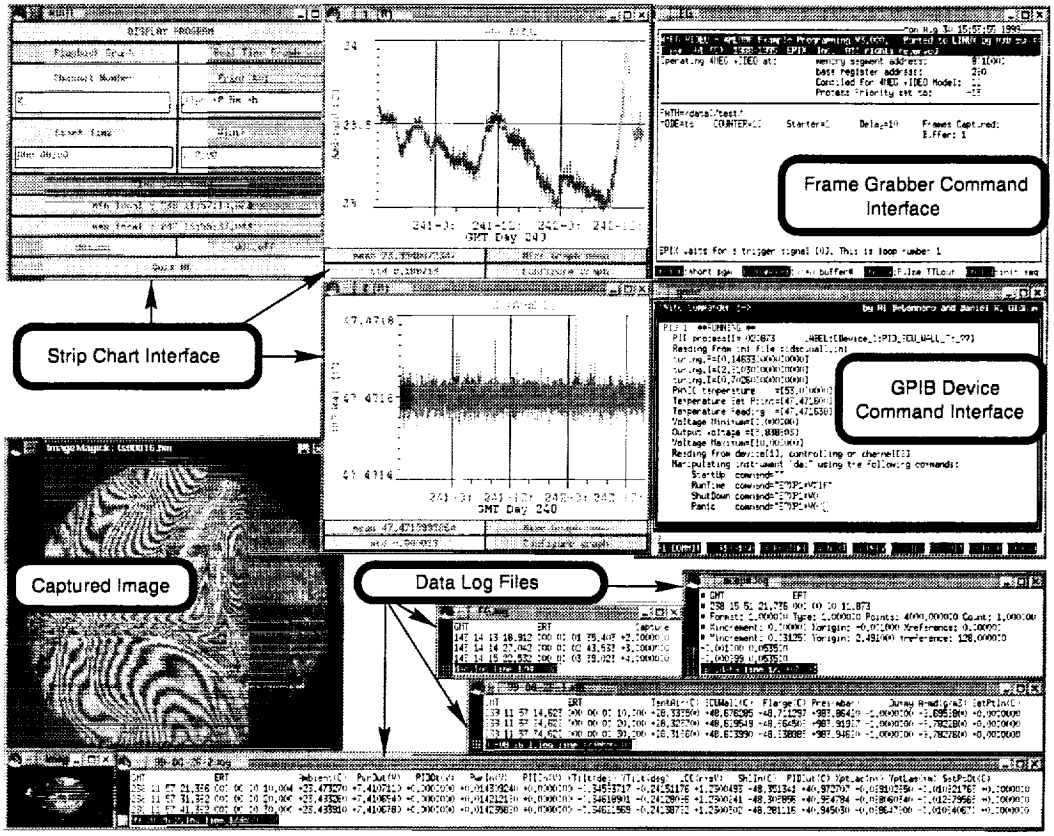

Figure 5: Screenshot of typical experiment runtime command and control.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	February 2000	Technical Memorandum

4. TITLE AND SUBTITLE

Remote Control and Data Acquisition: A Case Study

5. FUNDING NUMBERS

WU-101-53-00-00

6. AUTHOR(S)

Alfred J. DeGennaro and R. Allen Wilkinson

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
John H. Glenn Research Center at Lewis Field
Cleveland, Ohio 44135-3191

8. PERFORMING ORGANIZATION REPORT NUMBER

E-11963

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

NASA TM—2000-209634

11. SUPPLEMENTARY NOTES

Alfred J. DeGennaro, Cleveland State University, Cleveland, Ohio 44115, and R. Allen Wilkinson, NASA Glenn Research Center. Responsible person, R. Allen Wilkinson, organization code 6712, (216) 433-2075.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Category: 61 Distribution: Nonstandard

This publication is available from the NASA Center for AeroSpace Information, (301) 621-0390.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

This paper details software tools developed to remotely command experimental apparatus, and to acquire and visualize the associated data in soft real time. The work was undertaken because commercial products failed to meet the needs. This work has identified six key factors intrinsic to development of quality research laboratory software. Capabilities include access to all new instrument functions without any programming or dependence on others to write drivers or virtual instruments, simple full screen text-based experiment configuration and control user interface, months of continuous experiment run-times, order of 1% CPU load for condensed matter physics experiment described here, very little imposition of software tool choices on remote users, and total remote control from anywhere in the world over the Internet or from home on a 56 Kb modem as if the user is sitting in the laboratory. This work yielded a set of simple robust tools that are highly reliable, resource conserving, extensible, and versatile, with a uniform simple interface.

14. SUBJECT TERMS

Virtual instruments; Realtime; Data acquisition; LINUX; Telescience; Remote control; Data visualization

15. NUMBER OF PAGES

22

16. PRICE CODE

A03

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	